Java - Classes e Métodos

BCC 221 - Programação Orientada a Objectos(POO)

Guillermo Cámara-Chávez

Departamento de Computação - UFOP

Construtores

- Java também utiliza construtores para inicializar objetos assim que são criados
  - Por padrão, o compilador fornece um construtor *default*, que não possui argumentos e inicializa os membros de um objeto
    - Tipos primitivos são zerados;
    - O construtor de objetos internos são chamados automaticamente.

Construtores (cont.)

- O operador **new** invoca o construtor;
- A chamada do construtor é indicada pelo nome da classe seguida de parênteses
  - O construtor deve ter o mesmo nome da classe;
  - Não retorna nada e não possui tipo.

Construtores (cont.)

```
public class GradeBook{
    private String courseName;

    // construtor
    public GradeBook(String name){
        setCourseName(name);
    }

    // método para definir o nome da disciplina
    public void setCourseName( String name ){
        courseName = name;
    }
```

Construtores (cont.)

```
    // método para recuperar o nome da disciplina
    public String getCourseName() {
        return courseName;
    }

    // mostra a mensagem de bem vindas
    public void displayMessage() {
        // mostra a mensagem
        System.out.printf("Welcome to the grade book
          for\n%s!\n", getCourseName() );
    }
} // fim classe GradeBook
```

Construtores (cont.)

```
public class GradeBookTest{
    // método principal que inicia a execução
    public static void main( String[] args )
    {
        // cria um objeto da classe GradeBook
        GradeBook myGradeBook = new GradeBook("BCC221");

        System.out.printf( "Initial course name is:
          %s\n\n", myGradeBook.getCourseName() );
```

Exemplo

Construir uma classe *Conta* que matém o *saldo* de uma conta bancária além do *nome* do cliente.

Exemplo (cont.)

```
public class Conta {
    private String nome;
    private double saldo;

    public Conta(){
    }

    public Conta(String nome, double saldo){
        setNome(nome);
        setSaldo(saldo);

    }

    public void setNome(String nome){
        this.nome = nome;
    }

    public String getNome(){
        return nome;
    }
```

Exemplo (cont.)

```
    public void setSaldo(double saldo){
        this.saldo = saldo > 0 ? saldo : 0;
    }

    public double getSaldo(){
        return saldo;
    }

    public void deposita(double valor){
        double tmp = getSaldo();
        setSaldo(tmp + valor);
    }

    public void print(){
        System.out.println("Nome : " + getNome() + "\
            nSaldo: " + getSaldo());
    }
}
```

Exemplo (cont.)

```
public class DriverConta {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        Conta c1 = new Conta();
        Conta c2 = new Conta("Leonardo", 1500.56);

        System.out.print("Digite nome: ");
        String nome = input.nextLine();
        System.out.print("Digite saldo: ");
        double saldo = input.nextDouble();

        c1.setNome(nome);
        c1.setSaldo(saldo);

        c1.print();
        c2.print();
    }
}
```

Finalizadores e Coleta de Lixo Automática

- Toda classe em Java deriva da classe **Object**;
- Um dos métodos herdados é o **finalize**
  - Raramente utilizado;
  - Dos 6500 códigos da API Java, somente 50 o utilizam.

Finalizadores e Coleta de Lixo Automática (cont.)

- Cada objeto criado consome recursos de sistema, como memória
  - A JVM realiza a coleta de lixo automática (*garbage collection*);
  - Quando não há mais referências a um objeto na memória, tal objeto é marcado para a coleta de lixo;
  - Quando o coletor de lixo (*garbage collector*) for executado, os recursos utilizados por aquele objeto serão liberados.

Finalizadores e Coleta de Lixo Automática (cont.)

- Desta forma, estouros de memórias, comuns em C e C++, são menos propensas a ocorrer
- O método *finalize* é **chamado pelo coletor de lixo** para realizar a **terminação do objeto** antes que seus recursos sejam liberados;

Finalizadores e Coleta de Lixo Automática (cont.)

- Um problema com o método *finalize* é que não há garantia de que o coletor de lixo será executado antes de o programa terminar
  - Logo, a execução de sua aplicação não deve depender dele.
- De fato, desenvolvedores profissionais indicam que o método *finalize* não é útil em aplicações Java para empresas
  - Outras técnicas de liberação de recursos devem ser utilizadas;
  - Geralmente, as classes relacionadas à manipulação de recursos providenciam outras maneiras de liberá-los

Membros *static*

- Assim como em C++, o modificador static define membros que serão instanciados uma única vez
  - Ou seja, não haverá uma cópia para cada objeto;
  - Todos os objetos compartilham uma única cópia.
- O exemplo a seguir mostra a utilização de membros *static* e também do método *finalize*.

Membros *static* (cont.)

```
public class Rec {
    private static int n = 0;
    public Rec(){
        n++;
    }
    protected void finalize(){
        n--;
        System.out.println("Finalizou um objeto");
    }
    public static int getRec(){
        return n;
    }
}
```

Membros *static* (cont.)

```
public class JavaStatic {

    public static void main(String[] args) {

        System.out.println("Variavel estatica: "
                           + Rec.getRec());
        Rec r1 = new Rec();
        Rec r2 = new Rec();
        Rec r3 = new Rec();
        System.out.println("Variavel estatica: "
                           + Rec.getRec());
        r1 = null; r2 = null; r3 = null;
        System.gc();
        System.out.println("Variavel estatica: "
                           + Rec.getRec());

    }

}
```

Membros *static* (cont.)

```
Variavel estatica: 0
Variavel estatica: 3
Variavel estatica: 3
Finalizou um objeto
Finalizou um objeto
Finalizou um objeto
```

Membros *static* (cont.)

```
protected void finalize() throws Throwable
{
    try{
        // código de finalização
        } finally{
            super.finalize();
        }
}
```

Membros *static* (cont.)

- Se uma variável *static* não for inicializada, o compilador atribui um valor padrão;
- O método *static gc* do pacote *System* faz uma chamada explícita ao coletor de lixo
  - Não necessariamente todos os objetos serão coletados.

Membros *static* (cont.)

- Um método *static* não pode acessar membros não *static*
  - Um método *static* pode ser chamado mesmo quando não houver um objeto instanciado;
  - Pelo mesmo motivo, o ponteiro *this* não pode ser utilizado em um método *static*.
- Um método *static* deve ser invocado pelo nome da classe seguido de **.** ou por um objeto.

Geração de Números Aleatórios

- Existe duas formas de gerar número aleatórios
  - A classe *Random* gera números inteiros, reais em diferentes intervalos
  - O método estático *Math.random()* gera números reais entre 0 e 1.

Geração de Números Aleatórios (cont.)

- Classe Random
  - *nextDouble()*: número real [0,1]
  - *nextFloat()*: número real [0,1]
  - *nextGaussian()*: número com distribuição normal (média = 0, desvio padrão = 1)
  - *nextInt(int n)*: números interior [0, n[

Geração de Números Aleatórios (cont.)

Gerar 10000 números aleatórios e determinar a frequencia de cada um;

Geração de Números Aleatórios (cont.)

```
import java.util.*;

public class ExemploRandom {

    public static void myRandomMath(int m_ini, int
        m_final){
        int tot = (m_final - m_ini) + 1;
        int n;
        int hist[] = new int[tot];
        for (int i = 0; i < 100000; i++){
            n  = (int)(Math.random()*(m_final-m_ini+1)
                + m_ini) ;
            hist[n-m_ini]++;
        }
        print(hist);
    }
```

Geração de Números Aleatórios (cont.)

```
public static void myRandomClass(int m_ini, int
    m_final){
    int tot = (m_final - m_ini) + 1;
    int n;
    int hist[] = new int[tot];
    Random numRandom = new Random();
    for (int i = 0; i < 100000; i++){
        n  = numRandom.nextInt(tot);
        hist[n]++;
    }
    print(hist);
}
```

Geração de Números Aleatórios (cont.)

```
    public static void print(int[] v){
        for (int elem : v)
            System.out.println(elem + " ");
    }

    public static void main(String[] args) {
        System.out.println("Frequencia com Math.random"
            );
        myRandomMath(5, 10);
        System.out.println("Frequencia com Random class
            ");
        myRandomClass(5, 10);
    }

}
```

Geração de Números Aleatórios (cont.)

```
Frequencia com Math.random
16673
16500
16818
16750
16729
16530
Frequencia com Random class
16789
16639
16702
16625
16420
16825
```

Exemplo II

Um jogo popular de azar é um jogo de dados conhecido como craps, que é jogado em cassinos e nas ruas de todo o mundo. As regras do jogo são simples e diretas:

1. Você lança dois dados. Cada dado tem seis faces que contêm um, dois, três, quatro, cinco e seis pontos, respectivamente.

2. Depois que os dados pararam de rolar, a soma dos pontos nas faces viradas para cima é calculada.

3. Se a soma for 7 ou 11 no primeiro lance, você ganha.

4. Se a soma for 2, 3 ou 12 no primeiro lance (chamado "craps"), você perde (isto é, a "casa" ganha).

Exemplo II (cont.)

5. Se a soma for 4, 5, 6, 8, 9 ou 10 no primeiro lance, essa soma torna-se sua "pontuação". Para ganhar, você deve continuar a rolar os dados até "fazer sua pontuação" (isto é, obter um valor igual à sua pontuação).

6. Você perde se obtiver um 7 antes de fazer sua pontuação.

Exemplo II (cont.)

```
public class Craps {
    private static final Random randomNumber = new
        Random();

    private enum Status {CONTINUE, WON, LOST};

    private static final int SNAKE_EYES = 2;
    private static final int TREY = 3;
    private static final int SEVEN = 7;
    private static final int YO_LEVEN = 11;
    private static final int BOX_CARDS = 12;

    public void play(){
        int myPoint = 0;
        Status gameStatus;

        int sumOfDice = rollDice();
```

Exemplo II (cont.)

```
switch(sumOfDice){
    case SEVEN:
    case YO_LEVEN:
        gameStatus = Status.WON;
        break;
    case SNAKE_EYES:
    case TREY:
    case BOX_CARDS:
        gameStatus = Status.LOST;
        break;
    default:
        gameStatus = Status.CONTINUE;
        myPoint = sumOfDice;
        System.out.println("Pontuacao : " +
            myPoint);
}
```

Exemplo II (cont.)

```
        while ( gameStatus == Status.CONTINUE) {
            sumOfDice = rollDice();

            if (sumOfDice == myPoint)
                gameStatus = Status.WON;
            else
                if (sumOfDice == SEVEN)
                    gameStatus = Status.LOST;
        }

        if (gameStatus == Status.WON){
            System.out.println("O jogador vence");
        }
        else{
            System.out.println("O jogador perde");
        }
    }
```

Exemplo II (cont.)

```
    public int rollDice(){
        int die1 = 1 + randomNumber.nextInt(6);
        int die2 = 1 + randomNumber.nextInt(6);

        int sum = die1 + die2;

        System.out.println("Jogador lanca :" + sum);
        return sum;
    }
}
```

Exemplo II (cont.)

```
package crapstest;

public class CrapsTest {

    public static void main(String[] args) {
        Craps jogo = new Craps();
        jogo.play();
    }

}
```

FIM